POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SMBUD Project - MongoDB

Author(s): **Gabriele Ginestroni**
Giacomo Gumiero
Lorenzo Iovine
Nicola Landini
Francesco Leone

Group Number: **10**

Academic Year: 2022-2023

Contents

1 | Introduction

In this chapter will be presented the problem specification and the hypothesis under which the database is implemented.

1.1. Problem Specification

This project aims to build a documental database that handles scientific articles contained in the DBLP bibliography. The focus is on creating a database which allows efficient information retrieval of the articles, including their chapters and images. The main collections analyzed in the project are *Authors* and *Publication* with all their attributes and related objects like: *chapters, biographies* and *images*.

1.2. Assumptions

1. Articles can be published on a single venue
2. There is no distinction between different types of Publication
3. An author can't work for more than one organization for the same Publication
4. A chapter can contain subsections and these will be considered chapters as well

2 | ER Diagram

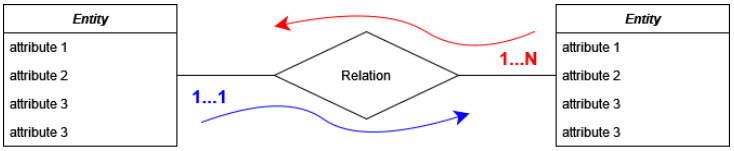


Figure 2.1: ER Diagram Organization

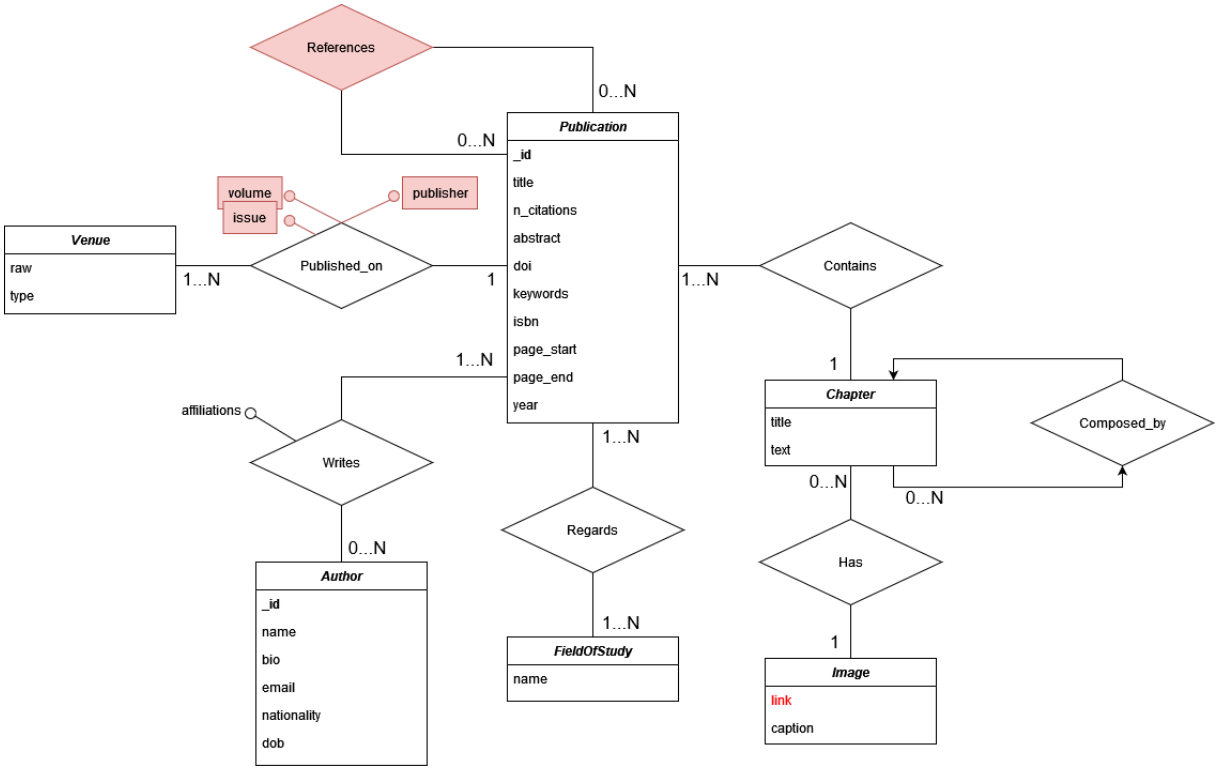


Figure 2.2: ER Diagram

Note: the primary keys of *Author* and *Publication* are written in bold. We decided not to highlight the primary keys of the other entities because they are not collections in our implementation.

In the end we have to say that, because we are aggregating *Venues* on raw field, we decided to move volume, issue and publisher from *Venue* to the relationship *Published_on*.

The Entity-Relationship model contains 6 main entities, that are related to each other through various relationships:

- **Publication:** represents all the scientific articles. Its attributes are: *_id, title, n_citations, abstract, doi, keywords, isbn, page_start, page_end, year* and its organization will be presented later
- **Author:** it is the one who contributed to a publication. Its attributes are: *_id, name, bio, email, nationality, dob (date of birth)*
- **Venue:** it is where a publication is published or presented. Its attributes are: *raw, type*
- **FieldOfStudy:** this entity represents the subjects of the publication and its attribute is *name*
- **Chapter:** it represents the chapter of a scientific articles. Its attributes are *title, text*
- **Image:** it represents the image contained in a specific chapter of the publication. Its attributes are *url, caption*

The ER diagram designed contains also the following relationships:

- **References:** is the relationship between a *publication* and another *publication* cited by the first one. This relationship also contains fields issue, volume and publisher
- **Published_on:** is the relationship between a *publication* and its *venue*
- **Writes:** is the relationship between *author* and *publication* which features the affiliation property. We decided to design it with `affiliation` as an attribute of the relationship, due to the fact that it belongs only to a pair of *author* and *publication* and it represents the institute where the author worked for the publication
- **Regards:** is the relationship between a *publication* and its *fields of study*
- **Contains:** is the relationship between *publication* and its *chapters*
- **Composed_by:** is the relationship between two *chapters*. It represents the relationship created between a chapter and its sections, between a section and its subsection and so on. Note that we used a directed arrow in order to indicate that a section belongs only to a chapter, but a chapter could own more than one section.

- **Has:** is the relationship between *chapter* and its *images*

3 | Document Structure

In this section we will describe the structure of our database. We decided to split our dataset in two collections: *authors* and *articles*. The reason behind this choice was to increase the performance and to reduce the spatial complexity. For example if we need to modify the email of an author we just have to change the field in the *author* collection. Whereas, if we just kept one collection, very expensive update query would have been needed.

Furthermore, to avoid redundancies we used manual references to bind the two collections.

In order to show the number of *Authors* and *Articles* in our collections, we performed the two following queries:

```
> db.articles.aggregate([
    { $group: {"_id": 1 , "count": { $sum: 1 }}} ])
< { _id: 1, count: 6381 }
> db.authors.aggregate([
    { $group: {"_id": 1 , "count": { $sum: 1 }}} ])
< { _id: 1, count: 17025 }
```

3.1. Data Preprocessing

Starting from the database sample used for the Neo4j part some operations were performed to add the required fields and to exploit better mongoDB functionalities.
Missing `isbn`, `page_start`, `page_end` attributes were added with random generations.
Most important, chapters were added from the original paper pdf when available, and parsing was performed by using PHP Library Smalot PDFParser. This library only parses the full text of the papers so the text was then divided into chapters using chapter titles as a reference to split, thus it was possible to obtain the 2 attributes' title and text

for each chapter (for the `text` field, only 10 rows were taken into consideration). A total of 170 pdfs were parsed and that was also used as a pool to pick random chapters for papers that could not be parsed because their pdfs were not available as a free copy on aminers' site.
To add images and captions, a dataset of films and their covers from tmdb was used, for every chapter a random number of images links were added combined with movie titles used as a caption.
The advantage of aminers' dataset was the built-in `_id` attribute for articles and authors and also the existence of an array of references with the list of the referenced `_id` inside every article tuple.
To better exploit this opportunity we believe it was handy to have the 2 collections of articles and authors, so we performed `JSON` parsing of the dump to generate 2 new `JSON` files:

- one that contains the articles, with an array of authors that has inside the references to their `_id` objectId
- the other contains the authors with an array of articles that has inside the references to their `_id` objectId.

In the authors' collection the bio attribute was randomly generated only when not available, while `email`, `nationality` and `dob` attributes were 100% randomly generated, especially:

- **email:** was generated by concatenating Name.Surname with a pool of most famous mail servers(gmail.com, yahoo.com...)
- **nationality:** was generated by picking randomly from a pool of 20 nation code
- **dob - date of birth:** was generated randomly from 1940 to 1992, so it may be possible that some inconsistencies are present (e.g. author that published before having 20 years old or even before being born)

3.2. Article structure

In the following `JSON` file we can see how the *Article* is structured in our dataset.
Note: in some fields we decided not to insert all the content only for ease of read reasons.
Note: in `venue`, fields `issue`, `volume` and `publisher` are missing because they are empty in this example.


```
{
"_id": {
  "$oid": "53e99f86b7602d9702859fdf"
},
"title": "Locality Sensitive Outlier Detection: A ranking driven
   approach",
"authors": [
  {
  "id": {
    "$oid": "542a4c9fdabfae61d496694e"
  },
  "org": "Computer Science and Engineering Department, OhioSU, USA"
  },
  {
  "id": {
    "$oid": "53f48bc5dabfaea7cd1cce1d"
  },
  "org": "Computer Science and Engineering Department, OhioSU, USA"
  },
  {
  "id": {
    "$oid": "53f44b6fdabfaec09f1dd00d"
  },
  "org": "Computer Science and Engineering Department, OhioSU, USA"
  }
],
"n_citation": 60,
"abstract": "Outlier detection is fundamental to a variety of database
   and...",
"doi": "10.1109/ICDE.2011.5767852",
"keywords": [
  "database point",
  "ranking scheme",
  "geometric approach",
  ...
],
"isbn": "978-1-4244-8958-9",
```

```
"page_start": "410",
"page_end": "421",
"year": 2011,
"fos": [
  "Locality-sensitive hashing",
  "Anomaly detection",
  "Machine learning",
  ...
],
"venue": {
  "raw": "ICDE",
  "type": 0,
},
"chapters": [
  {
  "title": "1. Introduction",
  "text": "Open Computing Language (OpenCL)[3] is a unified programming
    ...",
  "images": [
    {
    "caption": "Giraffada",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "A Certain Magical Index: The Miracle Of Endymion",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Ready To Rumble",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Raw Force",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "The Departed",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Repeaters",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    }
  ]
```

```
  },
  {
  "title": "2. The OpenCL Framework",
  "text": "21 Organization of Our Runtime\nThe target cluster...",
  "images": [
    {
    "caption": "The King Of New York",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "The Ghost Who Walks",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Fighter In The Wind",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Wake Up",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Infierno Blanco",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Bajo El Mismo Techo",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Le Dernier Samaritain",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    }
  ]
  },
  {
  "title": "3. Evaluation",
  "text": "We have implemented the OpenCL runtime and...",
  "images": [
    {
    "caption": "L'hotel Degli Amori Smarriti",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
```

```
    "caption": "Bit",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Dead Silence",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Amenazados",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    }
  ]
  },
  {
  "title": "4. Conclusions",
  "text": "We introduce the design and implementation of...",
  "images": [
    {
    "caption": "Eddie The Eagle",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Gorenos",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Pinocchio",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Le Grand Bazar",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Gangsters",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    },
    {
    "caption": "Kodachrome",
    "link": "https://image.tmdb.org/t/p/w600_and_h900_bestv2/....jpg"
    }
  ]
  }
],
```

```
"references": [
  {
    "$oid": "53e99fddb7602d97028b7e65"
  }
]
}
```

3.3. Author structure

In the following JSON file we can see how the *Author* is structured in our dataset.
Note: in some fields we decided not to insert all the content only for ease of read reasons.

```
{
"_id": {
  "$oid": "53f45775dabfaee4dc8162e6"
},
"name": "Guillermo Jorge-Botana",
"articles": [
  {
  "$oid": "53e99f86b7602d970285a187"
  }
],
"orcid": "0000-0001-5879-6783",
"bio": "Qing-Long Han received the B.Sc. degree in mathematics from the
    ...",
"email": "Guillermo.Jorge-Botana@yahoo.com",
"nationality": "de",
"dob": 1974-09-14T00:00:00.000+00:00
}
```

3.4. Attributes Description

In this section we will present all the attributes contained in our filtered dataset.

3.4.1. Article

Publication represent the central concept of the system and contains:

- **_id** is an ObjectId that identifies a publication.
- **title** represents the title of the publication.
- **authors** is an array of subdocuments that contains: ObjectId of the authors of the article and the `org` field which represent the affiliation.
- **n_citation** is the number of times that the publication has been mentioned.
- **abstract** is a string containing a brief summary of the contents of the paper.
- **doi** Digital Object Identifier is a persistent and standardized identifier.
- **keywords** is an array containing keywords of the publication.
- **isbn** is an identification code of the venue of the publication.
- **page_start** defines the starting page of the publication.
- **page_end** defines the last page of the publication.
- **year** represents the year of publication.
- **fos** is an array containing the fields of study of the publication.
- **venue** is a sub-document that represents where a publication is published or presented. This field contains:
  - **raw** is the name or the abbreviation of the venue (regardless the year, issue or volume) in which the publication was presented.
  - **type** indicates the type of the publication.
  - **volume** is the volume of the venue in which the article has been published.
  - **issue** refers to how many times a periodical has been published during that year.
  - **publisher** is the name of the publisher of the article.

- **chapters** is an array of sub-documents containing containing `title` (title of the chapter), `text` (the content), `sections` (that has the same structure of a chapter). They also contains `images` composed by:
  - **caption** the caption of the image.
  - **link** the link to the image.
- **references** set of ObjectIds of the referenced articles.

3.4.2. Author

The dataset provides the following author fields:

- **_id** is an ObjectId that identifies an author.
- **name** is the name of the author.
- **articles** is a set of articles identifier of the publications of the author.
- **orcid** Open Researcher and Contributor ID is a unique identifier for authors of scientific articles.
- **bio** is a string that describes the author.
- **email** is the email address of the author.
- **nationality** is the nationality of the author.
- **dob** is the birth date of the author.

4 | Commands and Queries

4.1. Commands

We have identified the following `INSERT` and `UPDATE` commands to show the system basic functionalities.

4.1.1. Insert a publication in the system

Assuming it is not present in the dataset, we inserted a new document that is a new instance of *Publication*. In order to do that we instantiated 5 different variables: `article_id` that generates an ObjectId representing the id of the article we're creating; `article_ref1` and `article_ref2` that are the ids of two scientific articles cited by this publication; `author1_id` and `author2_id` that are the ids of the authors.
Note: `isbn` field is missing
Note: `type` = 1 represents a *Journal*
Note: `issue` field is missing

```
article_id = ObjectId()
article_ref1 = ObjectId("53e99fe4b7602d97028bf743")
article_ref2 = ObjectId("53e99fddb7602d97028bc085")
author1_id = ObjectId()
author2_id = ObjectId()

db.articles.insertOne({
  _id: article_id,
  title: "An extensive study of C-SMOTE, a Continuous Synthetic Minority
    Oversampling Technique for Evolving Data Streams",
  authors:
  [
    {id:author1_id, org:"Politecnico di Milano"},
    {id:author2_id, org:"Politecnico di Milano"}
  ],
  n_citation: 3,
```

```
16  abstract: "Streaming Machine Learning (SML) studies algorithms that
     update their models, given an unbounded and often non-stationary flow
     of data performing a single pass. Online class imbalance learning is
     a branch of SML that combines the challenges of both class imbalance
     and concept drift. In this paper, we investigate the binary
    classification problem by rebalancing an imbalanced stream of data in
     the presence of concept drift, accessing one sample at a time.",
17  doi: "10.1016/j.eswa.2022.116630",
18  keywords: ["Evolving Data Stream","Streaming","Concept drift","
     Balancing"],
19  page_start: 39,
20  page_end: 46,
21  year: 2022,
22  fos: ["Computer Science","Stream Reasoning","Big Data"],
23  venue: {
24    raw: "ESA",
25    type: 1,
26    volume: 196,
27    publisher: "Elsevier"
28  },
29  chapters:
30  [
31    {
32    title:"1. Introduction",
33    text:"Nowadays, data abound as a multitude of smart devices, such as
     smartphones, wearables, computers, and Internet of Things (IoT)
    sensors produce massive, continuous, and unbounded flows of data,
    namely data streams. This poses several challenges to Machine
    Learning (ML)."
34    },
35
36    {
37    title:"2. Background",
38    text:"This section is divided into three parts describing the
    different concept drift types and characteristics, the evaluation
    metrics, and the most common approaches to use in class imbalance.",
39     subsection: [
40       {
41       title:"2.1. Concept drift in evolving data streams",
42       text:"In this part, we introduce the concept drift phenomenon
    explaining why and how it happens. We explain all its different types
    , forms, and possible speeds of occurrence.",
43       subsubsection: [
44         {
```

```
45        title:"2.1.1. Concept drift types",
46        text:"Since the generating function is unknown, concept drift is
     unpredictable. In the batch settings, with all the data available,
    it is simple to check and detect if a dataset is not stationary.",
47        images:
48          [
49            {
50            caption:"Fig.1 Representation of the three different types
    of concept drift",
51            url:"https://ars.els-cdn.com/content/image/1-s2.0-
    S0957417422001208-gr1.jpg"
52            }
53          ]
54        }
55      ]
56      }
57    ]
58    },
59
60    {
61    title:"3. C-SMOTE",
62    text:"This section recalls the description of C-SMOTE, inspired by
    the Smote technique, originally presented in Bernardo, Gomes et al.
    (2020). C-SMOTE is designed to rebalance an imbalanced data stream,
    and it can be pipelined with any SML- model. C-SMOTE stands for
    Continuous-Smote, meaning that the new Smote version is applied
    continuously.",
63    images:[
64      {
65      caption:"Fig. 2. Architecture of C-SMOTE meta-strategy pipelined
    with an Online Learner.",
66      url: "https://ars.els-cdn.com/content/image/1-s2.0-
    S0957417422001208-gr5.jpg"
67      }
68    ],
69    subsection:[
70      {
71      title:"3.1. Artificial data streams",
72      text: "To synthetically reproduce the different types of concept
    drifts shown in Section 2.1, we choose two of the most commonly used
    artificial data generators: SINE1 (Gama et al., 2004) and SEA (Street
     & Kim, 2001)."
73      }
74    ]
```

```
75
76      }
77      ],
78    references: [
79      article_ref1,
80      article_ref2
81    ]
82
83  })
```

4.1.2. Insert an author in the system

Assuming he is not present in the dataset, we used `insertOne` to create a new instance of *Author*.

Note: `author1_id` and `article_id`, refers to the variables instantiated in the previous command (Section: 4.1.1)

```
1  db.authors.insertOne({
2    _id: author1_id,
3    name: "Emanuele Della Valle",
4    orcid: "0000-0002-5176-5885",
5    articles:[
6      article_id
7    ],
8    bio:"Emanuele Della Valle holds a PhD in Computer Science from the
       Vrije Universiteit Amsterdam and a Master degree in Computer Science
       and Engineering from Politecnico di Milano. He is associate professor
        at the Department of Electronics, Information and Bioengineering of
       the Politecnico di Milano.",
9    email:"emanuele.dellavalle@gmail.com",
10   nationality:"it",
11   dob:ISODate("1975-03-07T00:00:00.000Z")
12 })
```

4.1.3. Update the number of citations of referenced publications

With the following snippet of code is possible to increment the `n_citations` field of the *Publications* referenced by the article created in section 4.1.1.

Note: in this command we used `updateMany` in order to update both the referenced publications; `update` wasn't enough because only one of the two matching document would have been updated

```
db.articles.updateMany(
  { $or:[{_id:{$eq:ObjectId("53e99fe4b7602d97028bf743")}}, {_id:{$eq:
   ObjectId("53e99fddb7602d97028bc085")}}]},
  { $inc: { n_citation: 1} }
)
```

4.1.4. Modification of the biography of an author

This command allows to access *Authors* by field `name` in order to append a string to field `bio`

```
db.authors.updateMany(
  {name: "Emanuele Della Valle"},
  [{ $set: { bio: { $concat: [ "$bio", "He recently become full
   professot at ETH Zurich." ] } } }]
)
```

4.1.5. Add a publication to its author

This command allows to add the new publication (*Publication* creation presented in section 4.1.1), to one of its *Authors*.

Note: we assumed `newArticleId` as the identifier of the new publication and that `author1_id` refers to the variable instantiated in section 4.1.1

```
newArticleId = ObjectId()

db.authors.updateOne(
  { _id:author1_id },
  { $push:{articles:newArticleId}}
)
```

4.2. Queries

We have identified the following queries in order to show the system's basic functionalities. For ease of read reasons, we decided to show results obtained using `project` operator, and, in some cases, we showed only some of the results.

4.2.1. Query 1

This query returns one publication written after 2013 whose `FieldOfStudy(fos)` contains *'Machine learning'*.

```
db.articles.findOne({
  "$and": [{year: {$gte:2013}}, {"fos":"Machine learning"}]
})
```

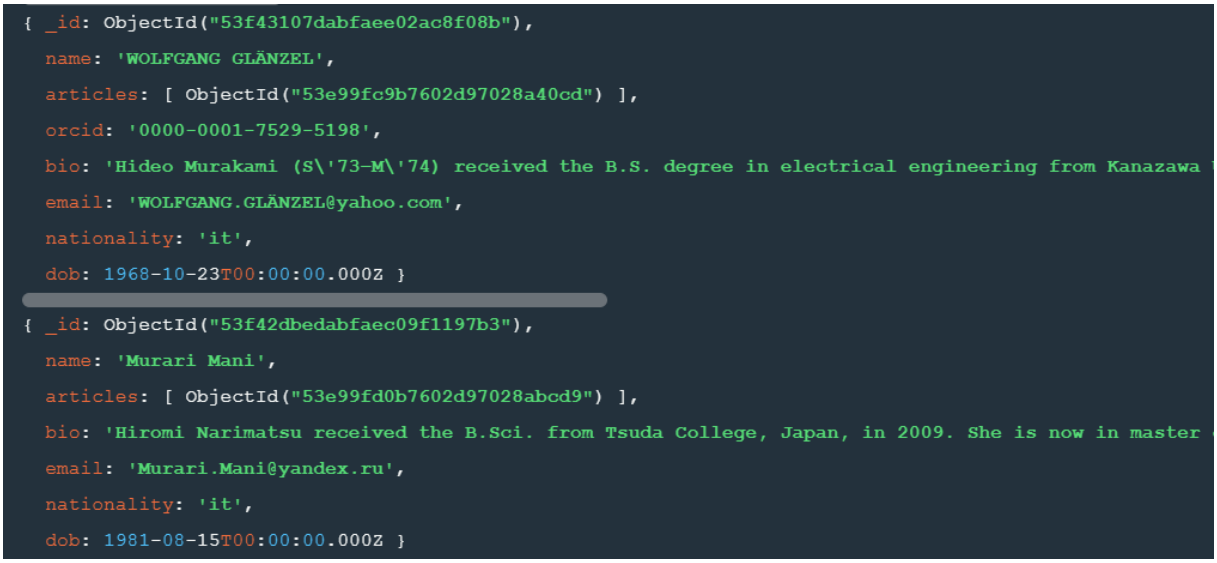

```
{ _id: ObjectId("53e99f86b7602d970285ab95"),
  title: 'Bootstrapping polarity classifiers with rule-based classification',
  year: 2013,
  fos:
   [ 'Bag-of-words model',
     'Rule-based system',
     'Bootstrapping',
     'Computer science',
     'Artificial intelligence',
     'Classifier (UML)',
     'Margin classifier',
     'Classifier (linguistics)',
     'Linear classifier',
     'Machine learning',
     'Quadratic classifier' ],
  chapters:
   [ { title: '1. Offline Learning of Multiple Regression Model' },
     { title: '2. Analysis on Online Time Complexity' },
     { title: '3. Analysis on Offline Time Complexity' } ] }
```


Figure 4.1: Projection on title, fos, year, chapters.title

4.2.2. Query 2

This query returns all the Italian authors born after 1960.

```
db.authors.find({
  $and:[{nationality:'it'},{dob:{$gte:ISODate("1960-01-01")}}]
})
```



4.2.3. Query 3

This query returns ten of the articles published by *'Elsevier'* after 2009.

Description: this query perform a filtering over the year attribute of the article and on the publisher field of the venue sub-document.

```
db.articles.find({
  $and:[{"venue.publisher":'Elsevier'},{year:{$gte:2009}}]
}).limit(10)
```

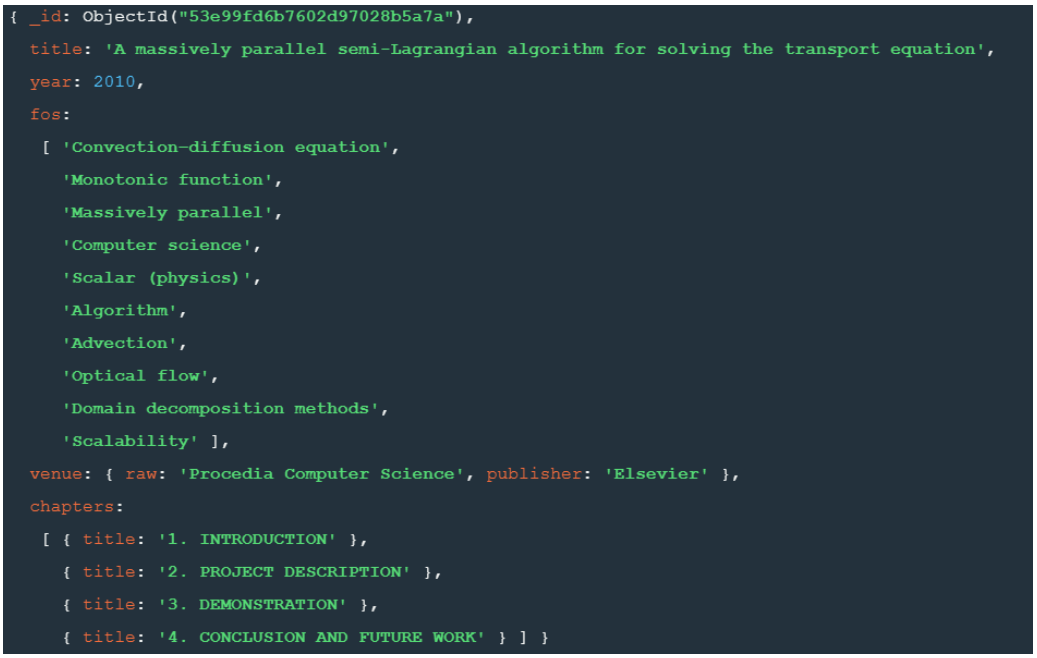
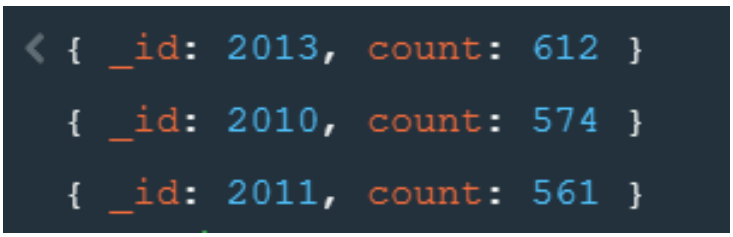


Figure 4.2: Projection on title, fos, year, chapters.title, venue.raw, venue.publisher

4.2.4. Query 4

This query returns the top three years sorted by number of publications.
Description: the query computes the count by aggregating with respect to the year field. Then groups are sorted by descending order and only the top 3 are kept.

```
db.articles.aggregate([
  {"$group" : {_id:"$year", count:{$sum:1}}},
  {"$sort": {count:-1}},
  {"$limit": 3}
]);
```



4.2.5. Query 5

This query finds all the articles with at least one Stanford affiliation and regarding *'Machine learning'* field of study.

Description: this query perform a filtering over the fos attribute of the article and on the org field of the authors array of sub-documents.

```
db.articles.aggregate([
  {$match: {"authors.org":{$regex: "Stanford"}}},
  {$match: {"fos":"Machine learning"}}
])
```

```
{ _id: ObjectId("53e99fe4b7602d97028c5012"),
  title: 'Individualizing generic decision models using assessments as evidence',
  authors:
   [ { id: ObjectId("53f45e83dabfaee02ad76294"),
       org: 'Department of Medicine, University of California, San Diego, CA' },
     { id: ObjectId("53f4cb99dabfaeeb14f811bc"),
       org: 'Management Science and Engineering Department, Stanford University, Stanford, CA' } ],
  fos:
   [ 'Decision analysis',
     'Decision rule',
     'Data mining',
     'Decision tree',
     'Optimal decision',
     'Computer science',
     'Artificial intelligence',
     'Decision model',
     'Decision field theory',
     'Decision quality',
     'Evidential reasoning approach',
     'Machine learning' ] }
```

Figure 4.3: Projection on authors, fos and title

4.2.6. Query 6

This query returns the top three years sorted by number of distinct authors.

Description: the query starts by unwinding the authors array. Then, articles are grouped by year and by author. The distinct count over the year is performed grouping the previous results by year and accumulating 1 for each group. Finally, results are sorted by descending order, keeping only the top 3 years.

```
db.articles.aggregate([
  {$unwind: "$authors"},
  {$group: {"_id":{ "year":"$year", "author":"$authors.id"}}},
  {$group: {"_id": "$_id.year", "count": { $sum: 1 }}},
  {$sort : {"count" : -1}},
  {$limit : 3 }
]);
```

```
{ _id: 2013, count: 2016 }
{ _id: 2010, count: 1823 }
{ _id: 2011, count: 1750 }
```

4.2.7. Query 7

This query returns the 20 most frequent `keywords`.

Description: the query starts by unwinding the keywords array. The next stage groups with respect to the keywords (case insensitive) and computes the count for each group. Finally, the results are ordered by descending order and limited to the top 20

```
db.articles.aggregate([
  {$unwind: "$keywords" },
  {
    $group: {
      _id: {$toLower: '$keywords'},
      count: {$sum: 1}
    }
  },
  {$sort : { count : -1}},
  {$limit : 20}
]);
```

```
{ _id: 'data mining', count: 167 }
{ _id: 'internet', count: 116 }
{ _id: 'computer science', count: 114 }
{ _id: 'information retrieval', count: 100 }
{ _id: 'feature extraction', count: 98 }
{ _id: 'real time', count: 93 }
{ _id: 'algorithm design and analysis', count: 86 }
{ _id: 'neural network', count: 85 }
{ _id: 'indexing terms', count: 83 }
{ _id: 'computational complexity', count: 81 }
{ _id: 'computer architecture', count: 78 }
{ _id: 'hardware', count: 74 }
{ _id: 'mathematical model', count: 73 }
{ _id: 'optimization', count: 73 }
{ _id: 'computational modeling', count: 72 }
{ _id: 'satisfiability', count: 72 }
{ _id: 'machine learning', count: 72 }
{ _id: 'protocols', count: 72 }
{ _id: 'testing', count: 69 }
{ _id: 'application software', count: 67 }
```

4.2.8. Query 8

This query, given the title of an article, returns the chapter with the highest number of images.

Description: the first stage matches the article given the title. Then an unwind of the chapters array is performed. The project stage is used to compute the variable `imgCount` which stores the number of images contained in each chapter. The results are sorted by descending order and limited to top 1.

```
db.articles.aggregate([
  {"$match" : {title: "Locality Sensitive Outlier Detection: A ranking
   driven approach"}},
  {"$unwind" : {path: "$chapters"}},
  {"$project": {
    "title":1,
    "chapters":1,
    "imgCount": { "$size": "$chapters.images" } } },
  {"$sort": {imgCount:-1}},
  {"$limit": 1}
]);
```


```
{ _id: ObjectId("53e99f86b7602d9702859fdf"),
  title: 'Locality Sensitive Outlier Detection: A ranking driven approach',
  chapters:
   [ { title: '2. The OpenCL Framework',
       text: '21 Organization of Our Runtime\nThe target cluster architecture consists of a single host node and\nmultiple compute nodes
       images:
        [ { caption: 'The King Of New York',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/OxtxltJK6N2aNMb6x4shSdiFwY.jpg' },
          { caption: 'The Ghost Who Walks',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/nCJq5mzrzuekS2g0BLc3Q2aYMIQ.jpg' },
          { caption: 'Fighter In The Wind',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/hQsnDNn99NDbCvyj9SLxj3hBAkH.jpg' },
          { caption: 'Wake Up',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/wv9CJql799ej93urPm5NQScHxrC.jpg' },
          { caption: 'Infierno Blanco',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/7IBGsaUXMol2K8I1BTBgxEBuS6y.jpg' },
          { caption: 'Bajo El Mismo Techo',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/5SRD0sZ4AuU89L1VuxecD9W2ck2.jpg' },
          { caption: 'Le Dernier Samaritain',
            link: 'https://image.tmdb.org/t/p/w600_and_h900_bestv2/b0wLwVkWgff3088oPyW6fcCzkhh.jpg' } ] },
  imgCount: 7 }
```


4.2.9. Query 9

This query returns articles citing another article that contains, between its `references`, at least one article written in a Milan University.

Description: the first stage loads, by id, the array of cited documents into a new field

called refs. Then a match is used to find in this field all the articles that contain at least 1 author that worked for a Milan university.

Note: we used the operator `lookup` to issue a join, in order to access another article instance.

```
db.articles.aggregate([
  {$lookup: {
    from: "articles",
    localField: "references",
    foreignField: "_id",
    as: "refs"
    }
  },
  {$match: {
    "refs.authors.org":{$regex: "Milan"}
    }
  },
  {"$project": {
    "title":1,
    "refs.title":1,
    "refs.authors": 1
    }
  }
])
```

```
{ _id: ObjectId("53e9a3abb7602d9702cba703"),
  title: 'Adapt cases: extending use cases for adaptive systems',
  refs:
   [ { title: 'Live goals for adaptive service compositions',
       authors:
        [ { id: ObjectId("53f46a7adabfaec09f24fcb5"),
            org: 'Politecnico di Milano, Milano, Italy' },
          { id: ObjectId("53f43983dabfaedce554c415"),
            org: 'Politecnico di Milano, Milano, Italy' } ] } ] }
```

Figure 4.4: Projection on title, refs.title, refs.authors

4.2.10. Query 10

This query finds the author with the maximum amount of written articles and retrieves his article with the highest number of `coauthors`.

Description: First of all, we compute the `artCount` field which contains the length of

the articles array field. Then we sort in descending order and keep only the first author by `artCount`. After that, a join is performed to load the articles documents into the new field `articles_doc`. A new projection is then used to compute the number of authors of each loaded article. Finally we sort the result in descending order and keep only the top 1 article. The projection is used to display the result in a clearer way.

Note: we used the operator `lookup` to issue a join in order to access articles of the author.

```
db.authors.aggregate([
  {"$project": {
    "_id":1,
    "name":1,
    "articles":1,
    "artCount": { "$size": "$articles" } } },
  {$sort : {"artCount": -1}},
  {$limit :1 },
  {$lookup: {
    from: "articles",
    localField: "articles",
    foreignField: "_id",
    as: "articles_doc"}},
  {"$unwind" : {path: "$articles_doc"}},
  {"$project": {
    "name":1,
    "articles_doc":1,
    "artCount":1,
    "authCount": { "$size": "$articles_doc.authors"}}},
  {$sort : { "authCount": -1}},
  {$limit : 1},
  {"$project": {
    "name":1,
    "articles_doc.title":1,
    "articles_doc.authors":1,
    "authCount": 1,
    "artCount":1 }}
])
```

```
{ _id: ObjectId("53f48041dabfae963d25910a"),
  name: 'Qiang Yang',
  artCount: 6,
  articles_doc:
   { title: 'Heterogeneous transfer learning for image clustering via the social web',
     authors:
      [ { id: ObjectId("53f48041dabfae963d25910a"),
          org: 'Hong Kong University of Science and Technology, Kowloon, Hong Kong' },
        { id: ObjectId("53f46144dabfaefedbb75169"),
          org: 'Shanghai Jiao Tong University, Shanghai, China' },
        { id: ObjectId("53f48c90dabfaea7cd1cf446"),
          org: 'Shanghai Jiao Tong University, Shanghai, China' },
        { id: ObjectId("53f386a8dabfae4b34a1729d"),
          org: 'Shanghai Jiao Tong University, Shanghai, China' },
        { id: ObjectId("53f48c9bdabfaea7cd1cf6cc"),
          org: 'Shanghai Jiao Tong University, Shanghai, China' } ] },
  authCount: 5 }
```


Figure 4.5: Projection on name, articles_doc.title, articles_doc.authors, authCount, artCount

4.2.11. Query 11

This query returns all the articles written by authors whose names combined have all 26 letters of the alphabet.

Desciption: The query is done with the following steps:

- an unwind on articles is performed to obtain an entry for each article written by an author
- results are sorted by authors' name in order to have the list sorted alphabetically for further steps
- for each article, a group is performed and the authors list is pushed into an array
- a string concatenating all the authors names retrieved by article documents, is computed and converted to lowercase
- a map operation is performed to split letter by letter the entries; a filter with a regex to keep only alphabet characters is applied, then an unwind on letters is made
- entries are then grouped by article `_id` and authors list; letters are reduced into an array, then the size of this array is computed and only entries containing all 26 letters are matched
- finally, a join operation is performed on articles to obtain the title

```
db.authors.aggregate([
  {
    "$unwind" : {path: "$articles"}
  },
  { "$sort" : { "name": 1 }},
  {
    "$group" : {
      _id: "$articles",
      authors: {
        $push: {
          $concat: ["$name"]
        }
      }
    },
  },
  {
    $project: {
      "_id": 1,
      "authors": "$authors",
      "fullNames": {
        $reduce: {
          input: "$authors",
          initialValue: "",
          in: { $toLower: {$concat : ["$$value", "$$this"]}}
        }
      }
    }
  },
  {
    $project: {
      "_id": 1,
      "authors": "$authors",
      letters: {
        $filter: {
          input: {
            $map: {
              input: {
                $range: [ 0, { "$strLenCP": "$fullNames" } ]
              },
              in: {
                "$substrCP":
                [
                  "$fullNames",
                  "$$this",
```

```
                  1
                ]
              }
            }
          },
          cond: {
            $regexMatch: {
              input: "$$this",
              regex: '[a-z]'
            }
          }
        }
      }
    }
  },
  { $unwind: '$letters'},
  {
    $group: {
      _id: {
        "_id": '$_id',
        authors: "$authors"
      },
      letters: { $addToSet: '$letters' },
    }
  },
  {
    $project: {
      "_id" : 1,
      "authors": "$authors",
      "differentLetters": {
        "$size": "$letters"
      }
    }
  },
  { $match : {"differentLetters" : 26}},
  {
    $lookup: {
      from: "articles",
      localField: "_id._id",
      foreignField: "_id",
      as: "articles_doc"
    }
  },
  {
```

```
    $project: {
      "_id": "$_id._id",
      "title": "$articles_doc.title",
      "authors": "$_id.authors",
      "differentLetters": "$differentLetters"
    }
  }
])
```

```
{ _id: ObjectId("53e9a042b7602d9702929e98"),
  title: [ 'Building an information retrieval test collection for spontaneous conversational speech' ],
  authors:
   [ 'Bhuvana Ramabhadran',
     'Dagobert Soergel',
     'David S. Doermann',
     'Douglas W. Oard',
     'G. Craig Murray',
     'James Mayfield',
     'Jianqiang Wang',
     'Liliya Kharevych',
     'Martin Franz',
     'Samuel Gustman',
     'Stephanie Strassel',
     'Xiaoli Huang' ],
  differentLetters: 26 }
{ _id: ObjectId("53e9a202b7602d9702b01751"),
  title: [ 'Design principles for developing stream processing applications' ],
  authors:
   [ 'Bugra Gedik',
     'Chitra Venkatramani',
     'Deepak S. Turaga',
     'Henrique Andrade',
     'Jeffrey David Harris',
     'John Cox',
     'Olivier Verscheure',
     'Paul Jones',
     'William Szewczyk' ],
  differentLetters: 26 }
```

Figure 4.6: Projection on _id, title, authors, differentLetters

4.2.12. Improving Queries Perfomance

In the queries, we performed string searches on relatively small textual fields using the *$regex* operator. This kind of search is easy to use and works very well on small datasets, but it is not optimal in large databases as it is not utilizing indexes efficiently. If we

wanted to perform advanced and high-performing full-text search queries, we would have had to define textual indexes among the two collections. Since only one textual index can be created over a collection, an index over multiple textual fields should be defined. These indexes can require some disk space and use a lot of resources when created.

5 | Conclusions

The documental approach turned out to be very flexible and intuitive, thanks to its affinity with the object-oriented paradigm. Also, this kind of technology allowed us to shape data to match the most frequent operations that could take place in the publication domain. Therefore, queries became very simple and efficient.
Moreover, with our implementation, we tried to reach a good trade-off between performance and spatial complexity, avoiding choices that could lead to critical data duplication problems.